探索編 ミニマックス法

Phase3

フェーズの目標

このフェーズでは、評価関数（Phase 2）と組み合わせて、数手先の未来まで読み込む**探索アルゴリズム**を実装します。これにより、目の前の利得だけでなく、相手の最善の応手も考慮に入れた、より戦略的なAIが完成します。

主な実装対象は**ミニマックス法（ Minimax Algorithm）**です。

ミニマックス法の基礎構造

ミニマックス法は、

AI（Maximizer）が**評価値を最大化**する手を選び、

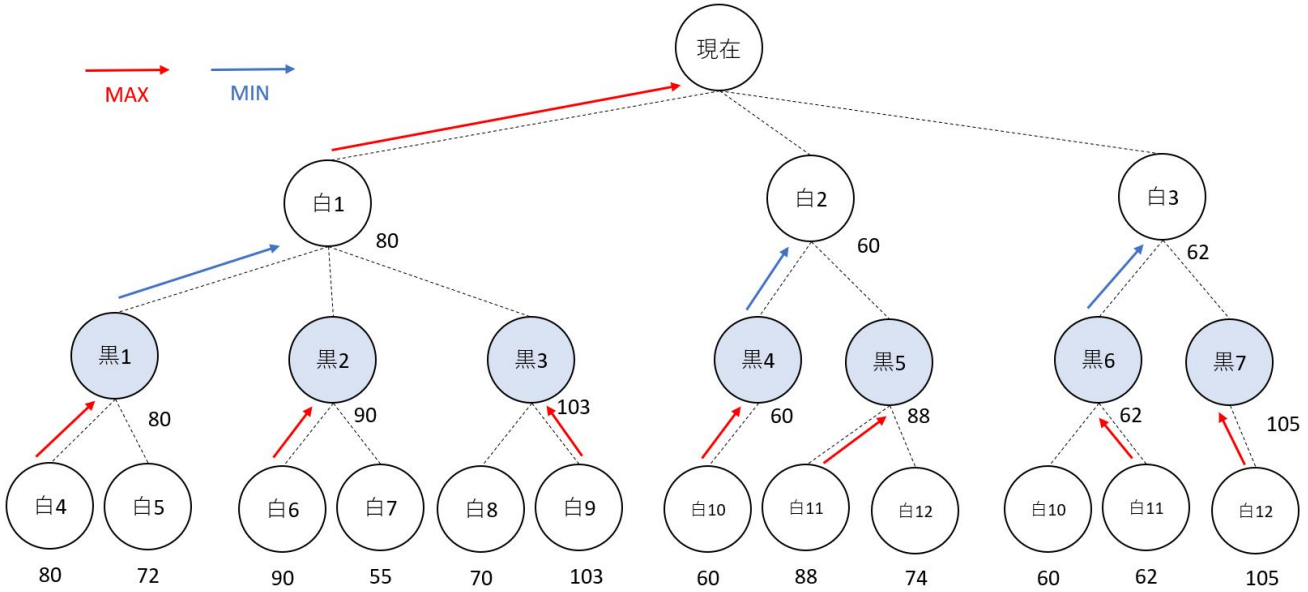
相手（Minimizer）が**評価値を最小化**する手を選ぶと仮定して、

木構造（ゲームツリー）を探索します。

例：白番で3手先まで読む場合

評価関数を使うのは一番先の手だけ。

そこから上は自分の番ではMaxを選び、相手の番ではMinを選ぶ。



再帰関数を使おう

```
// 評価を返す再帰関数
private int minimax(Board board, int depth, boolean isMaximizingPlayer) {
    // 略

  int eval = minimax(childBoard, depth - 1,  !isMaximizingPlayer);
```

depth(階層)が0になるまで自信(minmax関数)を呼び出す。

Max値をとる階層か、Min値をとる階層かをisMaximizingPlayerという引数で指定する。

実装のサンプル
（自信のある人はサンプルを見ずに実装してみよう)

think()の実装のサンプル

```
public String think(Board currentBoard) {
    // 探索の深さを設定（例: 4手先）
    final int SEARCH_DEPTH = 4;

    int myColor = currentBoard.getMyColor();
    List<String> legalMoves = currentBoard.getLegalMoves(myColor);
    if (legalMoves.isEmpty())
        return "pass";

    int maxScore = Integer.MIN_VALUE;
    List<String> bestMove = null;
    for (String move : legalMoves) {
        // moveを打った後の「仮のBoard」を作成
        Board newBoard = currentBoard.applyMoveAndClone(move);

        // 続く
```

think()の実装のサンプル（続き）

```
            // 仮のBoardを評価する
            // ２手目は相手のターンなので、最後の引数isMaximizingPlayerはfalseで呼び出す。
            int score = minmax(newBoard, SEARCH_DEPTH - 1, false);

            // １手目は自分のターンなので評価が大きい方を取る。
            if (maxScore < score) {
                maxScore = score;
                bestMove = new ArrayList<String>();
                bestMove.add(move);
            } else if (maxScore == score) {
                if (bestMove == null) System.out.println("[ERROR] score =" + score);
                bestMove.add(move);
            }
        }
        if (bestMove.size() == 1) {
            return bestMove.get(0);
        }

        // ベストが２つ以上ある場合はランダムに決める
        int index = random.nextInt(bestMove.size());
        return bestMove.get(index);
    }
```

minmax()の実装のサンプル

```
// 評価を返す再帰関数
private int minimax(Board board, int depth, boolean isMaximizingPlayer) {
    // 終了条件: 深さ0 または ゲーム終了
    if (depth == 0 || board.isTerminal()) {
        return evaluateBoard(board);
    }
    if (isMaximizingPlayer) {
        // Maximizer（自分）のターン: 評価値の最大化を目指す
        List<String> moves = board.getLegalMoves(board.getMyColor());
        if (moves.isEmpty()) {
            // 相手の合法手もなければ、両者パスでゲーム終了
            if (board.getLegalMoves(board.getOpponentColor()).isEmpty()) {
                return evaluateBoard(board); // 終端状態として評価値を返す
            }
            // 片方のみパスの場合、深さを減らさずに次のプレイヤーに手番を渡す
            return minmax(board, depth, false);
        }
        int maxScore = Integer.MIN_VALUE;
        for (String move : moves) {
            Board childBoard = board.applyMoveAndClone(move);
            int score = minmax(childBoard, depth - 1, false); // 次はMinimizer へ
            maxScore = Math.max(maxScore, score);
        }
        return maxScore;
    } else {
        // Minimizer（相手）のターン: 評価値の最小化を目指す
    }
}
```